# Platform SDK Developer's Guide

## Log4j2 Configuration with the Application Template Application Block

12/12/2025

# Contents

# Log4j2 Configuration with the Application Template Application Block

Platform SDK Application Template Application Block contains Log4j2 plugin component for logging configuration.

Its goal is to allow application to configure logging in accordance to the Common Genesys Logging Options with Log4j2 library.

## Enabling and configuring PSDK log4j2 logging

### Using AppTemplate Configuration Manager

GFApplicationConfigurationManager, once initialized, automatically applies and listen for updates in the application logging configuration.

```
public class MyApplication {
    protected static final LmsEventLogger LOG =
LmsLoggerFactory.getLogger(MyApplication.class);

...
    GFApplicationConfigurationManager appManager =
            GFApplicationConfigurationManager.newBuilder()
            .withCSEndpoint(new Endpoint("CS-primary", csHost1, csPort1))
            .withCSEndpoint(new Endpoint("CS-backup", csHost2, csPort2))
            .withClientId(clientType, clientName)
            .withUserId(csUsername, csPassword)
            .build();
    appManager.register(new GFAppCfgOptionsEventListener() {
        public void handle(final GFAppCfgEvent event) {
            Log.getLogger(getClass()).info("The application configuration options received: "
+ event);
            // Init or update own application options from 'event.getAppConfig()'
        }});
    appManager.init();

    // LmsEventLogger method usage:
    LOG.log(CommonLmsEnum.GCTI_APP_INIT_COMPLETED);
    // Common ILogger method usage:
    LOG.info("Some Log4j2 info message");
...
```

For more details see Additional Logging Features.

### Using AppTemplate API calls

If GApplicationConfigurationManager is not used, its still possible to configure logging in the same way by using AppTemplate API calls.

In package "com.genesyslab.platform.apptemplate.log4j2" AppTemplate provides classes named PsdkLog4j2Configuration and Log4j2Configurator.

Main cases are following:

1. Create PsdkLog4j2Configuration with its factory method "parse(KeyValueCollection)" and set it with the configurator (Log4j2Configurator.setConfig()). It uses optional config profile and given application Options "log" sections, and configures. Drawbacks: no "log-extended" Options section is taken into account, no Message Server appender from the application connections.
2. Obtain IGApplicationConfiguration and apply it with the configurator (Log4j2Configurator.applyLoggingConfig()). The application configuration may be explicitely filled using GApplicationConfiguration, or automatically initialized from ConfService using COMGApplicationConfiguration. This way includes initialization of "log-extended" Options sections and may add Message Server appender if its properly configured in the application configuration. Note: this configuration appliance method also switches PSDK loggers to send their logs to Log4j2 (if it was not enabled earlier).

More details on enabling of PSDK logging may be found in Setting up Logging in Platform SDK.

### Switch PSDK logging to log4j2 with system properties

The following properties are supported:

- "-Dcom.genesyslab.platform.commons.log.loggerFactory=log4j2" to switch PSDK commons logging implementation to send logs to log4j2.
- "-Dcom.genesyslab.platform.apptemplate.lmslogger.factory=log4j2" to switch AppTemplate LmsEventLogger's to send logs and LMS events to log4j2.

## AppTemplate Logging configuration profile

AppTemplate logging configuration profile is a usual Log4j2 xml configuration file. And it may be used as "startup configuration" before application is initialized and read its configuration from Configuration Server.

For details on Log4j2 configuration details see: http://logging.apache.org/log4j/2.x/manual/configuration.html.

To use "log4j2.xml" as the config profile, it should be referred with application CME Option:

```
[log]
"log4j2-config-profile"="log4j2.xml"
```

Example of startup logging configuration with logs formatted in the common Genesys LMS style:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration packages="com.genesyslab.platform.apptemplate.log4j2plugin">
  <Appenders>
    <GLogFile name="PSDKAppTpl-LogFile" fileName="startup.log">
      <GLmsLayout timeFormat="locale" messageFormat="full"/>
      <GLogRolloverStrategy expConfig="8 days"/>
    </GLogFile>
    <GLogFile name="PSDK" fileName="psdk.log">
```

```
      <GLmsLayout timeFormat="locale" messageFormat="full"/>
      <GLogRolloverStrategy expConfig="8 days"/>
    </GLogFile>
  </Appenders>
  <Loggers>
    <Root level="debug">
      <AppenderRef ref="PSDKAppTpl-LogFile"/>
    </Root>
    <Logger name="com.genesyslab.platform" level="debug">
      <AppenderRef ref="PSDK"/>
    </Logger>
  </Loggers>
</Configuration>
```

### Logging profile feature: "startup" appenders

Appenders, which have "PSDKAppTpl-" prefix in their names are treated as "startup" appenders.

Those ones are actual when the config profile is used as a usual log4j2 configuration file at the application startup time. But, when application has started, loaded its application configuration options, and called AppTemplate logging configuration (referring this profile), the "startup" appenders are excluded from the new configuration.

### Logging profile feature: separate subsystems logs

Specifics of the Genesys Logging Configuration Options is that log files may be generated and filtered by evel level, but not by logger or "source" name. I.e. generated by the "log" appenders are added to the "root" logger of logging configuration.

Logging configuration profile allows creation of dedicated appenders and custom loggers declaration for collecting of specific subsystem(s) logs separately from main application logs.

Example of such configuration is in the config profile sample above - appender named "PSDK", and logger named "com.genesyslab.platform". This file name is "hardcoded" in the config profile and can't be changed from application CME Options. But its still possible to change following parameters of the logger with "logger-<id>" option in "log-extended" section: "level" and "additivity" (see details bellow). By this way, its possible to change messages level of the PSDK log file, and to add or exclude PSDK messages from main application log files.

## Useful Platform SDK loggers for optional configuration

### PSDK root logger

Platform SDK loggers are named under PSDK base package - "com.genesyslab.platform". Usage of this logger name allows applications to separately configure, filter and record PSDK internal logs.

### Protocol messages tracing loggers

Usually, PSDK generates debug logs reflecting Protocols communications messages in truncated form together with other debug messages.

Sometimes applications may need to have separated recordings of protocols communications messages dumps.

For this purpose PSDK contains dedicated loggers:

- "com.genesyslab.platformmessage.request",
- "com.genesyslab.platformmessage.receive",
- and their common name - "com.genesyslab.platformmessage".

By default, these loggers are disabled. Its possible to enable them with jvm system property "-Dcom.genesyslab.platform.trace-messages=true".

> **Important**
>
> This system property is "branched" for support of values definitions for different protocol types. For example, "-Dcom.genesyslab.platform.Configuration.ConfServer.trace-messages=true" or "-Dcom.genesyslab.platform.Reporting.StatServer.trace-messages=true".

These values enable tracing for Config Server and Stat Server protocols only. "Branched" options names are constructed by insertion of a string representation of specific ProtocolDecription value (see DuplexChannel.getProtocolDescription().toString()).

It's possible to provide several different declarations for different protocols at a same time. It's also possible to enable tracing for the common property ("enable by default"), and disable for some specific protocols.

### ADDP logger

There is a dedicated specially named logger for ADDP traces - "com.genesyslab.platform.ADDP".

It's for possible adjustment of ADDP traces logging filtering.

## Application CME Logging Options

The AppTemplate logging configuration is based on AppTemplate Application Configuration structure as a snapshot representation of a COM AB CfgApplication structure.

Following values are taken into account: "log" and "log-extended" sections of the applications' "Options", and connected Message Server CME application.

### Options section "log"

Most of this section options are inherited from the Genesys Common Log Options descriptions, as listed in the Framework 8.5 Configuration Options Reference Manual.

### verbose

Specifies if log output is created, and if so, the minimum level of log events generated. Log event levels, starting with the highest priority level, are Standard, Interaction, Trace, and Debug.

**Default Value:** *all*

**Valid Values:**

- *all* - All log events (that is, log events of the Standard, Trace, Interaction, and Debug levels) are generated.
- *debug* - The same as all.
- *trace* - Log events of Trace level and higher (that is, log events of Standard, Interaction, and Trace levels) are generated, but log events of the Debug level are not generated.
- *interaction* - Log events of Interaction level and higher (that is, log events of Standard and Interaction levels) are generated, but log events of Trace and Debug levels are not generated.
- *standard* - Log events of Standard level are generated, but log events of Interaction, Trace, and Debug levels are not generated.
- *none* - No log output is produced.

**Changes Take Effect:** Immediately

**Additional Information:** Note: For definitions of the Standard, Interaction, Trace, and Debug log levels, refer to the Framework Management Layer User's Guide or Framework Genesys Administrator Help.

To configure log outputs, set log level options ("all", "alarm", "standard", "interaction", "trace", and/or "debug") to the desired types of log output ("stdout", "stderr", "network", "memory", and/or [filename], for log file output).

You can use:

- One log level option to specify different log outputs.
- One log output type for different log levels.
- Several log output types simultaneously, to log events of the same or different log levels.

You must separate the log output types by a comma when you are configuring more than one output for the same log level.

### all

Specifies the outputs to which an application sends all log events. The log output types must be separated by a comma when more than one output is configured.

**Default Value:** No default value.

**Valid Values:** Log output types:

- *stdout* - Log events are sent to the Standard output (stdout).
- *stderr* - Log events are sent to the Standard error output (stderr).
- *network* - Log events are sent to Message Server, which can reside anywhere on the network. Message Server stores the log events in the Log Database. Setting the all log level option to the network output enables an application to send log events of the *Standard*, *Interaction*, and *Trace* levels to Message Server. *Debug*-level log events are neither sent to Message

Server nor stored in the Log Database.

- [filename] - Log events are stored in a file with the specified name. If a path is not specified, the file is created in the application's working directory.

**Changes Take Effect:** Immediately

**Additional Information:** For example: `all = stdout, logfile`

Note1: The log output options are activated according to the setting of the Verbose configuration option.

Note2: To ease the troubleshooting process, consider using unique names for log files that different applications generate.

Warning! Directing log output to the console (by using "stdout", "stderr" settings) can affect application performance. Avoid using these log output settings in a production environment.

### alarm

Specifies the outputs to which an application sends the log events of the *Alarm* level. The log output types must be separated by a comma when more than one output is configured.

**Default Value:** No default value.

**Valid Values:** Log output types:

- *stdout* - Log events are sent to the Standard output (stdout).
- *stderr* - Log events are sent to the Standard error output (stderr).
- *network* - Log events are sent to Message Server, which can reside anywhere on the network. Message Server stores the log events in the Log Database.
- [filename] - Log events are stored in a file with the specified name. If a path is not specified, the file is created in the application's working directory.

**Changes Take Effect:** Immediately

**Additional Information:** For example: `alarm = stderr, network`

Note1: The log output options are activated according to the setting of the Verbose configuration option.

Note2: To ease the troubleshooting process, consider using unique names for log files that different applications generate.

Warning! Directing log output to the console (by using "stdout", "stderr" settings) can affect application performance. Avoid using these log output settings in a production environment.

### standard

Specifies the outputs to which an application sends the log events of the Standard level. The log output types must be separated by a comma when more than one output is configured.

**Default Value:** No default value.

**Valid Values:** Log output types:

- *stdout* - Log events are sent to the Standard output (stdout).
- *stderr* - Log events are sent to the Standard error output (stderr).
- *network* - Log events are sent to Message Server, which can reside anywhere on the network.

Message Server stores the log events in the Log Database.

- [filename] - Log events are stored in a file with the specified name. If a path is not specified, the file is created in the application's working directory.

**Changes Take Effect:** Immediately

**Additional Information:** For example: `standard = stdout, logfile`

Note1: The log output options are activated according to the setting of the Verbose configuration option.

Note2: To ease the troubleshooting process, consider using unique names for log files that different applications generate.

Warning! Directing log output to the console (by using "stdout", "stderr" settings) can affect application performance. Avoid using these log output settings in a production environment.

### interaction

Specifies the outputs to which an application sends the log events of the Interaction level and higher (that is, log events of the Standard and Interaction levels). The log outputs must be separated by a comma when more than one output is configured.

**Default Value:** No default value.

**Valid Values:** Log output types:

- *stdout* - Log events are sent to the Standard output (stdout).
- *stderr* - Log events are sent to the Standard error output (stderr).
- *network* - Log events are sent to Message Server, which can reside anywhere on the network. Message Server stores the log events in the Log Database.
- [filename] - Log events are stored in a file with the specified name. If a path is not specified, the file is created in the application's working directory.

**Changes Take Effect:** Immediately

**Additional Information:** For example: `interaction = stdout, logfile`

Note1: The log output options are activated according to the setting of the Verbose configuration option.

Note2: To ease the troubleshooting process, consider using unique names for log files that different applications generate.

Warning! Directing log output to the console (by using "stdout", "stderr" settings) can affect application performance. Avoid using these log output settings in a production environment.

### trace

Specifies the outputs to which an application sends the log events of the *Trace* level and higher (that is, log events of the *Standard*, *Interaction*, and *Trace* levels). The log outputs must be separated by a comma when more than one output is configured.

**Default Value:** No default value.

**Valid Values:** Log output types:

- *stdout* - Log events are sent to the Standard output (stdout).
- *stderr* - Log events are sent to the Standard error output (stderr).

- *network* - Log events are sent to Message Server, which can reside anywhere on the network. Message Server stores the log events in the Log Database. Setting the *all* log level option to the *network* output enables an application to send log events of the *Standard*, *Interaction*, and *Trace* levels to Message Server. *Debug*-level log events are neither sent to Message Server nor stored in the Log Database.
- [filename] - Log events are stored in a file with the specified name. If a path is not specified, the file is created in the application's working directory.

**Changes Take Effect:** Immediately

**Additional Information:** For example: `trace = stdout, logfile`

Note1: The log output options are activated according to the setting of the Verbose configuration option.

Note2: To ease the troubleshooting process, consider using unique names for log files that different applications generate.

Warning! Directing log output to the console (by using "stdout", "stderr" settings) can affect application performance. Avoid using these log output settings in a production environment.

### debug

Specifies the outputs to which an application sends the log events of the Debug level and higher (that is, log events of the *Standard*, *Interaction*, *Trace*, and *Debug* levels). The log output types must be separated by a comma when more than one output is configured.

**Default Value:** No default value.

**Valid Values:** Log output types:

- *stdout* - Log events are sent to the Standard output (stdout).
- *stderr* - Log events are sent to the Standard error output (stderr).
- *network* - Log events are sent to Message Server, which can reside anywhere on the network. Message Server stores the log events in the Log Database.
- [filename] - Log events are stored in a file with the specified name. If a path is not specified, the file is created in the application's working directory.

**Changes Take Effect:** Immediately

**Additional Information:** For example: `debug = stdout, logfile`

Note1: The log output options are activated according to the setting of the Verbose configuration option.

Note2: To ease the troubleshooting process, consider using unique names for log files that different applications generate.

Note3: Debug-level log events are never sent to Message Server or stored in the Log Database.

Warning! Directing log output to the console (by using "stdout", "stderr" settings) can affect application performance. Avoid using these log output settings in a production environment.

### segment

Specifies whether there is a segmentation limit for a log file. If there is, sets the mode of measurement, along with the maximum size. If the current log segment exceeds the size set by this option, the file is closed and a new one is created. This option is ignored if log output is not configured to be sent to a log file.

**Default Value:** 100 MB

**Valid Values:**

- false - No segmentation is allowed.
- <number>[ KB] - Sets the maximum segment size, in kilobytes. The minimum segment size is 100 KB.
- <number> MB - Sets the maximum segment size, in megabytes.
- <number> hr - Sets the number of hours for the segment to stay open. The minimum number is 1 hour.

**Changes Take Effect:** Immediately

### expire

Determines whether log files expire. If they do, sets the measurement for determining when they expire, along with the maximum number of files (segments) or days before the files are removed. This option is ignored if log output is not configured to be sent to a log file.

**Default Value:** 10

**Valid Values:**

- false - No expiration; all generated segments are stored.
- <number>[ file] - Sets the maximum number of log files to store. Specify a number from 1–1000.
- <number> day - Sets the maximum number of days before log files are deleted. Specify a number from 1–100.

**Changes Take Effect:** Immediately

**Additional Information:** Note: If an option’s value is set incorrectly (out of the range of valid values), it will be automatically reset to 10.

### compress-method

Platform SDK AppTemplate AB specific property to specify method that will be used for archiving log files.

The log option name is case insensitive, and can be "CompressMethod", "compress-method", or "compress_method".

**Default Value:** none

**Valid Values:**

- none - No archiving; all generated log files are stored "as-is".
- gzip - GZip archiving is to be used for "historical" log files.
- zip - Zip archiving is to be used for "historical" log files.
- zip<digit> - Zip archiving with given compression level is to be used for "historical" log files.

**Changes Take Effect:** Immediately

**Introduced in release:** 9.0.003.01

### time-convert

Specifies the system in which an application calculates the log record time when generating a log file. The time is converted from the time in seconds since "00:00:00 UTC, January 1, 1970".

The log option name is case insensitive, and can be: "TimeConvert", "time-convert", or "time_convert".

**Default Value:** local

**Valid Values:**

- local - The time of log record generation is expressed as a local time, based on the time zone and any seasonal adjustments. Time zone information of the application's host computer is used.
- utc - The time of log record generation is expressed as Coordinated Universal Time (UTC).

**Changes Take Effect:** Immediately

### time-format

Specifies how to represent, in a log file, the time when an application generates log records. A log record's time field in the ISO 8601 format looks like this: "2001-07-24T04:58:10.123".

The log option name is case insensitive, and can be: "TimeFormat", "time-format", or "time_format".

**Default Value:** time

**Valid Values:**

- time - The time string is formatted according to "HH:MM:SS.sss" (hours, minutes, seconds, and milliseconds) format.
- locale - The time string is formatted according to the system's locale.
- iso8601 - The date in the time string is formatted according to the ISO 8601 format. Fractional seconds are given in milliseconds.

**Changes Take Effect:** Immediately

### message-format

Specifies the format of log record headers that an application uses when writing logs in the log file. Using compressed log record headers improves application performance and reduces the log file's size. With the value set to short:

- A header of the log file or the log file segment contains information about the application (such as the application name, application type, host type, and time zone), whereas single log

records within the file or segment omit this information.

- A log message priority is abbreviated to Std, Int, Trc, or Dbg, for Standard, Interaction, Trace, or Debug messages, respectively.
- The message ID does not contain the prefix GCTI or the application type ID.

The log option name is case insensitive, and can be: "MessageFormat", "message-format", or "message_format".

**Default Value:** medium

**Valid Values:**

- short - An application uses compressed headers when writing log records in its log file.
- medium - An application uses medium size headers when writing log records in its log file.
- full - An application uses complete headers when writing log records in its log file.
- shortcsv - An application uses compressed headers with comma delimiter when writing log records in its log file. This value is only valid for Platform SDK Application Template-based applications.
- shorttsv - An application uses compressed headers with tab char delimiter when writing log records in its log file. This value is only valid for Platform SDK Application Template-based applications.
- shortdsv - An application uses compressed headers with message-header-delimiter delimiter when writing log records in its log file. This value is only valid for Platform SDK Application Template-based applications.
- custom - An application uses custom log messages format, which is separately defined in option custom-message-format or output-pattern. This value is only valid for Platform SDK Application Template-based applications.

**Changes Take Effect:** Immediately

**Additional Information:** A log record in the full format looks like this:

```
2002-05-07T18:11:38.196 Standard localhost cfg_dbserver GCTI-00-05060 Application started
```

A log record in the short format looks like this:

```
2002-05-07T18:15:33.952 Std 05060 Application started
```

Note: Whether the full, short, or any other format is used, time is printed in the format specified by the "time-format" option.

## use-native-levels

Platform SDK AppTemplate AB specific option.

It enables support of native log levels (like "Error", "Warn", etc) to be used for non-LMS messages in common LMS events formats instead of LMS levels like "Standard", "Interaction", etc.

The log option name is case insensitive, and can be: "UseNativeLevels", "use-native-levels", or "use_native_levels".

Note: This option is experimental and its value procession may get changed.

**Introduced in release:** 9.0.002.09

### message-header-delimiter

Platform SDK AppTemplate AB specific property as a parameter for "shortdsv" message format ("message-format").

The log option name is case insensitive, and can be: "MessageHeaderDelimiter", "message-header-delimiter", or "message_header_delimiter".

**Default Value:** |

### custom-message-format

Platform SDK AppTemplate AB specific option.

Value of this option is used as a log message pattern if "message-format" option value is equal to "custom".

In comparison with "output-pattern", this option provides predefined messages prefix containing timestamp (by "time-format"/"time-convert"), and the LMS-style log level.

The log option name is case insensitive, and can be: "CustomMessageFormat", "custom-message-format", or "custom_message_format".

Note: This option is experimental and its value procession may get changed.

**Introduced in release:** 9.0.002.01

### output-pattern

Platform SDK AppTemplate AB specific option.

Value of this option is used as a log message pattern if "message-format" option value is equal to "custom".

In comparison with "custom-message-format", this option does not provide predefined messages prefix like a timestamp with log level.

The log option name is case insensitive, and can be: "OutputPattern", "output-pattern", or "output_pattern".

**Introduced in release:** 9.0.002.05

### file-encoding

Platform SDK AppTemplate AB specific property for configuration of the log files encoding.

The log option name is case insensitive, and can be: "FileEncoding", "file-encoding", or "file_encoding".

**Default Value:** UTF-8

### file-header-provider

Platform SDK AppTemplate AB specific property for customization of the log files header content.

The log option name is case insensitive, and can be: "FileHeaderProvider", "file-header-provider", or "file_header_provider".

**Additional Information:** If this option is not specified, then the provider is taken with SPI declaration for the *com.genesyslab.platform.apptemplate.log4j2plugin.FileHeaderProvider* interface.

### enable-thread

Specifies whether to enable or disable the logging thread. If set to true (the logging thread is enabled), the logs are stored in an internal queue to be written to the specified output by a dedicated logging thread. This setting also enables the log throttling feature, which allows the verbose level to be dynamically reduced when a logging performance issue is detected.

The log option name is case insensitive, and can be: "EnableThread", "enable-thread", or "enable_thread".

**Default Value:** false

**Valid Values:** true, false

**Additional Information:** Refer to the Framework 8.5 Management Framework User's Guide for more information about the log throttling feature.

If this option is set to false (the logging thread is disabled), each log is written directly to the outputs by the thread that initiated the log request. This setting also disables the log throttling feature.

Note: Platform SDK AppTemplate AB does not support the log throttling feature.

### enable-location-for-thread

Platform SDK AppTemplate AB specific option to enable the call location information passing to the Log4j2 logging thread, which was enabled with option "enable-thread".

The log option name is case insensitive, and can be: "EnableLocationForThread", "enable-location-for-thread", or "enable_location_for_thread".

**Additional Information:** If one of the layouts is configured with a location-related attribute like HTML locationInfo, or one of the patterns %C or $class, %F or %file, %l or %location, %L or %line, %M or %method, Log4j will take a snapshot of the stack, and walk the stack trace to find the location information.

This is an expensive operation: 1.3 - 5 times slower for synchronous loggers. Synchronous loggers wait as long as possible before they take this stack snapshot. If no location is required, the snapshot will never be taken.

However, asynchronous loggers need to make this decision before passing the log message to another thread; the location information will be lost after that point. The performance impact of taking a stack trace snapshot is even higher

for asynchronous loggers: logging with location is 4 - 20 times slower than without location. For this reason, asynchronous loggers do not include location information by default.

### log4j2-config-profile

Platform SDK AppTemplate AB specific option.

It defines a base structure and may contain some predefined startup or permanent appenders, and initial custom loggers configurations.

The log option name is case insensitive, and can be: "Log4j2ConfigProfile", "log4j2-config-profile", or "log4j2_config_profile".

### default-logdir

Platform SDK AppTemplate AB specific option.

Default root directory for the log files. If specified, it is applied to file names defined in options like "standard", "interaction", "debug", etc.

It's used if log file name is not absolute path, and is not started from "./", or "../".

Note: This option value may be overridden with jvm system property "*default-logdir*".

The log option name is case insensitive, and can be: "DefaultLogdir", "default-logdir", or "default_logdir".

**Introduced in release:** 9.0.005.00

### msgsrv-intMsgsLevel

Platform SDK AppTemplate AB specific property to set log messages filter on Message Server Appender for Platform SDK internal events.

This value should not be lower than INFO level to not cause unlimited recursion/avalanche.

The log option name is case insensitive, and can be: "msgsrv-intMsgsLevel", "msgsrv_intMsgsLevel".

**Default Value:** info

**Valid Values:** info, warn, error, fatal, off

**Changes Take Effect:** Immediately

**Introduced in release:** 9.0.005.02

### message-file

Specifies the file name for application-specific log events. The name must be valid for the operating system on which the application is running. The option value can also contain the absolute path to the application-specific *.lms file. Otherwise, an application looks for the file in its working directory.

The log option name is case insensitive, and can be: "MessageFile", "message-file", or "message_file".

**Default Value:** As specified by a particular application

**Valid Values:** Any valid message file ("<filename>.lms")

**Changes Take Effect:** Immediately, if an application cannot find its "*.lms" file at startup

**Additional Information:** For example:

```
message-file = my-app.lms
```

Warning! An application that does not find its *.lms file at startup cannot generate application-specific log events and send them to Message Server.

### event-log-host

Platform SDK AppTemplate AB specific property to let user applications be able to override the applications' host name in log files and message server events.

It is used by the AppTemplate Log4j2 logging configuration functions in PsdkLog4j2Configuration and Log4j2Configurator.

The log option name (case insensitive): "EventLogHost", "event-log-host", or "event_log_host".

**Changes Take Effect:** Immediately

**Additional Information:** For example:

```
event-log-host = node-1-virtual-host
```

## Options section "log-extended"

Some of this section options are inherited from the Genesys Common Log Options descriptions, as listed in the Framework 8.5 Configuration Options Reference Manual.

### level-reassign-disable

When this option is set to true, the original (default) log level of all log events in the [log-extended] section are restored. This option is useful when you want to use the default levels, but not delete the customization statements.

**Default Value:** false

**Valid Values:** true, false

**Changes Take Effect:** Immediately

### level-reassign-<eventID>

Specifies a log level for log event <eventID> that is different than its default level, or disables log event <eventID> completely. If no value is specified, the log event retains its default level.

**Default Value:** Default value of log event <eventID>

**Changes Take Effect:** Immediately

**Additional Information:** This option is useful when you want to customize the log level for selected log events.

These options can be deactivated with the *level-reassign-disable* option.

See Genesys Common Log Options descriptions section of Framework 8.5 Configuration Options Reference Manual for more details on this option.

**Example:** this example shows customized log level settings, subject to the following log configuration:

```
[log]
verbose     = interaction
all         = stderr
interaction = log_file
standard    = network
```

Before the log levels of the log are changed:

- Log event 1020, with default level standard, is output to stderr and log_file, and sent to Message Server.
- Log event 2020, with default level standard, is output to stderr and log_file, and sent to Message Server.
- Log event 3020, with default level trace, is not generated.
- Log event 4020, with default level debug, is not generated.

Extended log configuration section:

```
[log-extended]
level-reassign-1020 = none
level-reassign-2020 = interaction
level-reassign-3020 = interaction
level-reassign-4020 = standard
```

After the log levels are changed:

- Log event 1020 is disabled and not logged.
- Log event 2020 is output to stderr and log_file.
- Log event 3020 is output to stderr and log_file.
- Log event 4020 is output to stderr and log_file, and sent to Message Server.

## logger-<id>

Platform SDK specific extended logging configuration option for applying of custom properties on loggers configuration.

**Changes Take Effect:** Immediately

**Additional Information:** The "log-extended" section of CME Application Options may contain multiple declarations of loggers customization records.

For example:

```
[log-extended]
logger-ID1 = "logger1.name: level=debug, additivity=false"
logger-ID2 = "logger2.name: level=error"
```

"ID1" and "ID2" are just unique identifiers of the logger declarations in the options section.

"logger1.name" and "logger2.name" are loggers names in a common log4j sense of loggers naming convention.

These values will be applied to the internally generated log4j2 configuration in the following way (the declarations will be created, or adjusted, if already exist):

```
  <Loggers>
    <Logger name="logger1.name" level="debug" additivity="false" />
    <Logger name="logger2.name" level="error" />
  </Loggers>
```

It's included in GAppLogExtOptions, which may contain list of such descriptions, and is a part of *GAppLoggingOptions*. The custom logger declaration supports following (optional) properties:

- level - the Level to be associated with the Logger;
- additivity ("true"/"false") - true if the logger should be additive, false otherwise;
- includeLocation ("true"/"false") - whether location should be passed downstream.